Introduction to Objective-C

Session 420

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Introduction to Objective-C

Michael Jurewitz
Developer Tools Evangelist

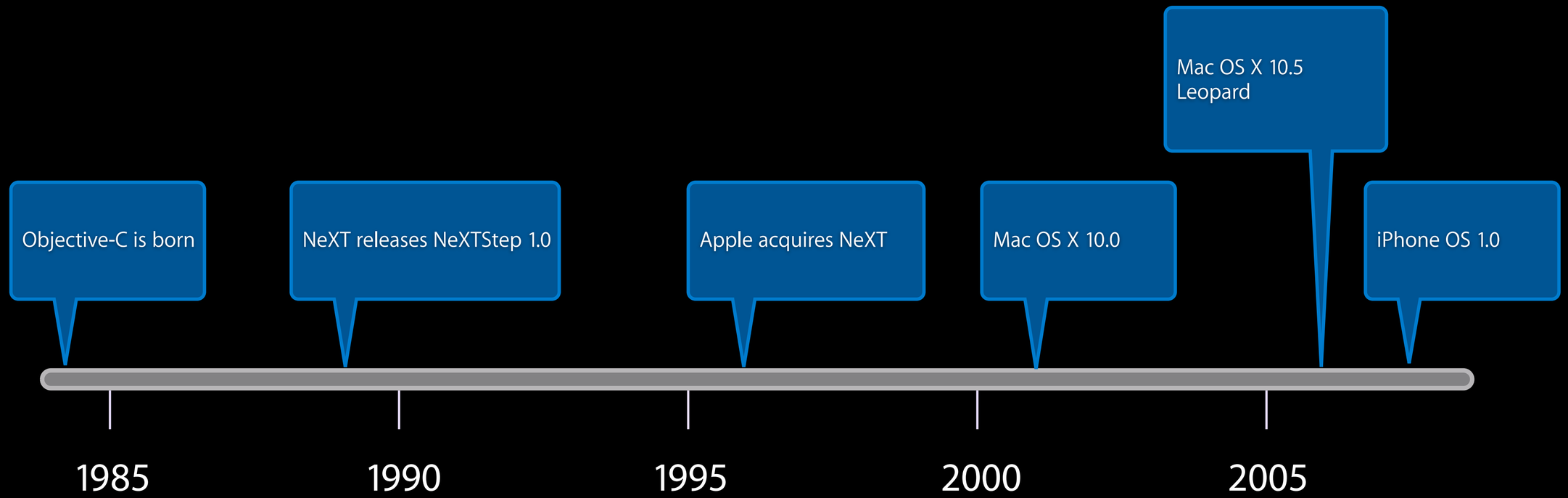
What You'll Get

- A working understanding of Objective-C
- Familiarity with important classes on Mac and iPhone
- Knowledge of common design patterns on Mac and iPhone

Objective-C

AT&T
9:42 AM
Text
Calendar
Photos
Camera
YouTube
Stocks
Maps
Weather
Clock
Calculator
Notes
Settings
iTunes
App Store
Phone
Mail
Safari
iPod

Objective-C


Objective-C is born
NeXT releases NeXTStep 1.0
Apple acquires NeXT
Mac OS X 10.0
Mac OS X 10.5
Leopard
iPhone OS 1.0
1985
1990
1995
2000
2005

Objective-C is born

NeXT releases NeXTStep 1.0

Apple acquires NeXT

Mac OS X 10.0

Mac OS X 10.5
Leopard

iPhone OS 1.0

25 years of evolution

1985 1990 1995 2000 2005

What Is Objective-C?

- Superset of C
- Dynamic method dispatch
- Runtime decision making
- Runtime class extensions

Calling a Method

A comparative example

C++:
```
someObject->doSomething();
```

Java:
```
someObject.doSomething();
```

Objective-C:
```
[someObject doSomething];
```

More Examples

```
[anEmployee salary];
```

Instance

Method

More Examples

```
[anEmployee salary];
```

```
[anEmployee setRaise:200];
```

Instance Method Argument

More Examples

```
[anEmployee salary];
```

```
[anEmployee setRaise:200];
```

```
[anEmployee setWork:work withDeadline:today];
```

More Examples

The difference is readability

```
new Glass(5,10,0.75);
```

```
[Glass glassWithWidth:5 height:10
      selfDestructProbability:0.75];
```

Working with Variables

```
Employee *anEmployee;
```

```
Employee *anEmployee = [Employee newEmployee];
```

```
Employee *anEmployee = [[Employee alloc] init];
```

Working with Variables

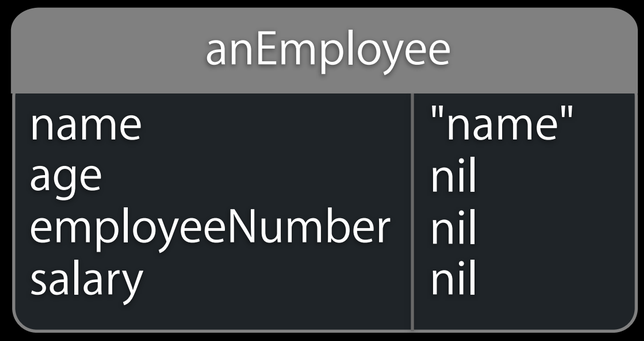
```
Employee *anEmployee = [[Employee alloc] init];
```

Working with Variables

```
Employee *anEmployee = [[Employee alloc] init];
```

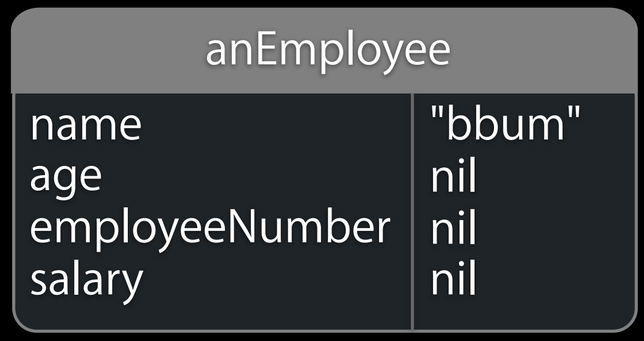
Working with Variables

```
Employee *anEmployee = [[Employee alloc] init];
```



Working with Variables

```
Employee *anEmployee = [[Employee alloc] initWithName:@"bbum"];
```



Objective-C Selectors

It's all in the name

- First class data type in Objective-C — SEL
- Basis for key design patterns in Cocoa
  - Delegation
  - Target/Action
  - NSThread
  - Many more…

Objective-C Selectors

Method	Selector
`[Employee salary]`	`@selector(salary)`
`[anEmployee setRaise:200]`	`@selector(setRaise:)`
`[anEmployee setWork:work withDeadline:today]`	`@selector(setWork:withDeadline:)`

Objective-C Selectors

Method	Selector
`[Employee salary]`	`@selector(salary)`
`[anEmployee setRaise:200]`	`@selector(setRaise:)`
`[anEmployee setWork:work withDeadline:today]`	`@selector(setWork:withDeadline:)`

Objective-C Selectors

Method	Selector
`[Employee salary]`	`@selector(salary)`
`[anEmployee setRaise:200]`	`@selector(setRaise:)`
`[anEmployee setWork:work withDeadline:today]`	`@selector(setWork:withDeadline:)`

Objective-C Selectors

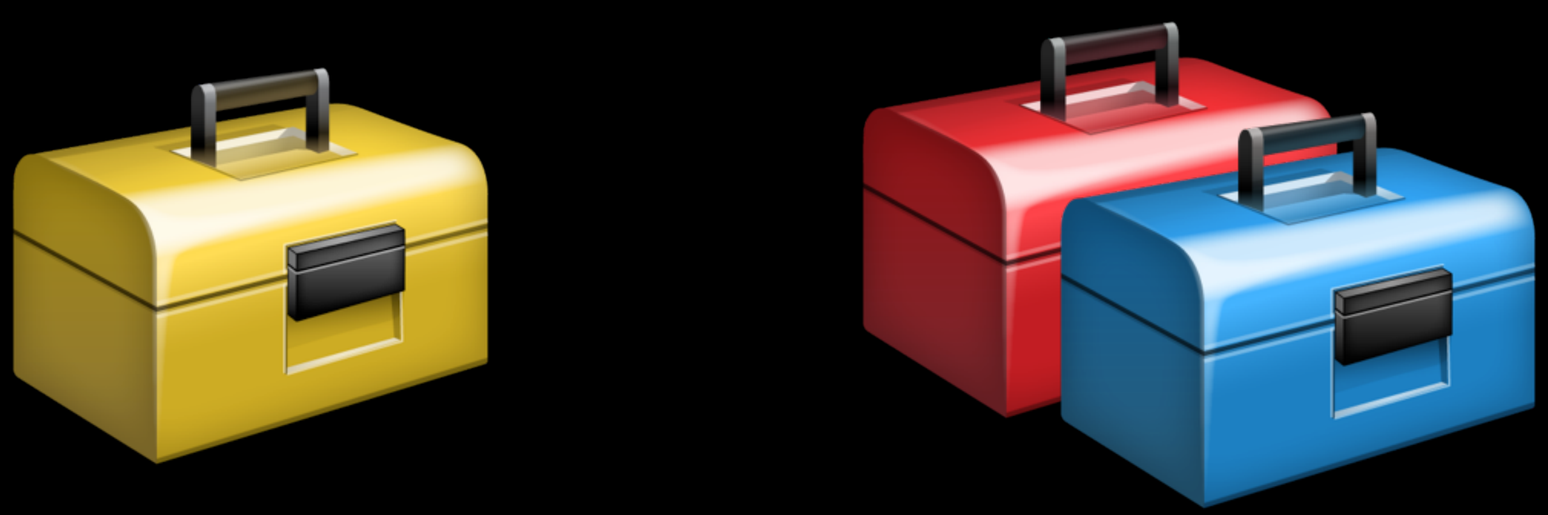
It's all in the name

```
[anEmployee respondsToSelector:@selector(doWork:)];
```

Frameworks

Frameworks

System-provided functionality



Foundation

AppKit/UIKit

Foundation

Foundation

Strings

```
NSString *myString = @"This is a String";
```

```
NSString *myString = [NSString stringWithFormat:@"WWDC%d", 2009];
```

```
NSString *pathString = @"Library/Caches/Images/dataCache.txt";
NSArray *pathComponents = [pathString pathComponents];
NSString *fileName = [pathString lastPathComponent];
```

```
NSString *fileContents = [NSString stringWithContentsOfFile:pathString];
```

Foundation

Arrays

- Mutable

```
NSMutableArray *arr = [NSMutableArray array];
[arr addObject:@"randomString"];
```

- Immutable

```
NSArray *arr = [NSArray arrayWithObjects:@"foo", @"bar",
                                         @"moof", nil];
NSString *complexString = [arr componentsJoinedByString:@"--"];
```

"foo--bar--moof"

Foundation

Dictionaries

- Mutable

```
NSMutableDictionary *md = [NSMutableDictionary dictionary]
[md setObject:@"someValue" forKey:@"someKey"];
```

- Immutable

```
NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:
    @"valueOne", @"keyOne", @"valueTwo", @"keyTwo", nil];
```

Foundation

Filesystem

- NSFileManager

```
NSFileManager *fm = [NSFileManager defaultManager];

if ([fm isWritableFileAtPath:path]) {
  [fm createFileAtPath:[path stringByAppendingPathComponent:@"foo.bar"]
                                                   contents:@"foo"
                                                 attributes:nil];
}
```

Foundation

Dynamic behavior

- Object introspection

```
if ( [anArchiver isKindOfClass:[NSCoder class]] )
```

```
if ( [anObject respondsToSelector:@selector(someMessage:)] )
```

- Dynamic method dispatch

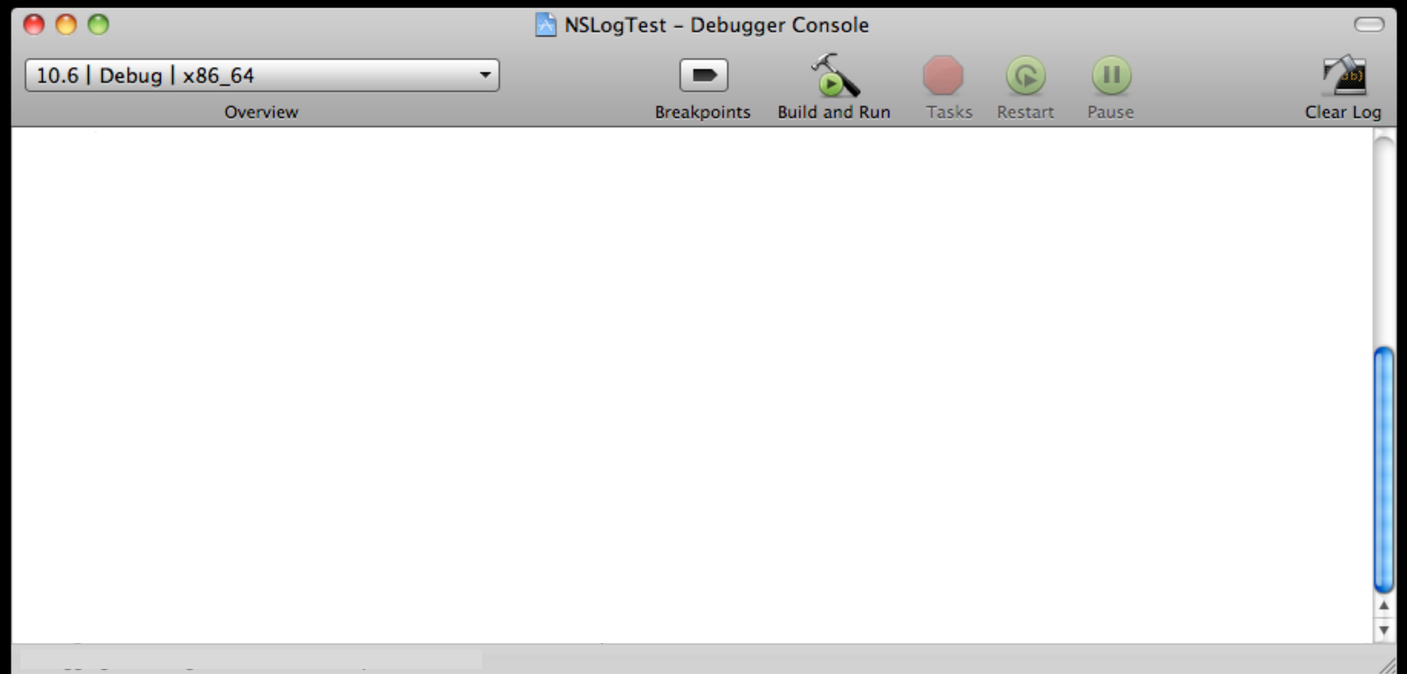
```
[anObject performSelector:@selector(goToWork)];
```

```
[anObject performSelectorInBackground:@selector(someWork:)
                           withObject:dataToProcess];
```

Foundation

NSLog

```
NSLog(@"Simple debug statement");

NSLog(@"There are over %d sessions at WWDC", numberOfSessions);

NSLog(@"Got an array: %@", myArray);
NSLog(@"Got a dictionary: %@", myDictionary);
```



Foundation

NSLog

```
NSLog(@"Simple debug statement");

NSLog(@"There are over %d sessions at WWDC", numberOfSessions);

NSLog(@"Got an array: %@", myArray);
NSLog(@"Got a dictionary: %@", myDictionary);
```

Foundation

NSLog

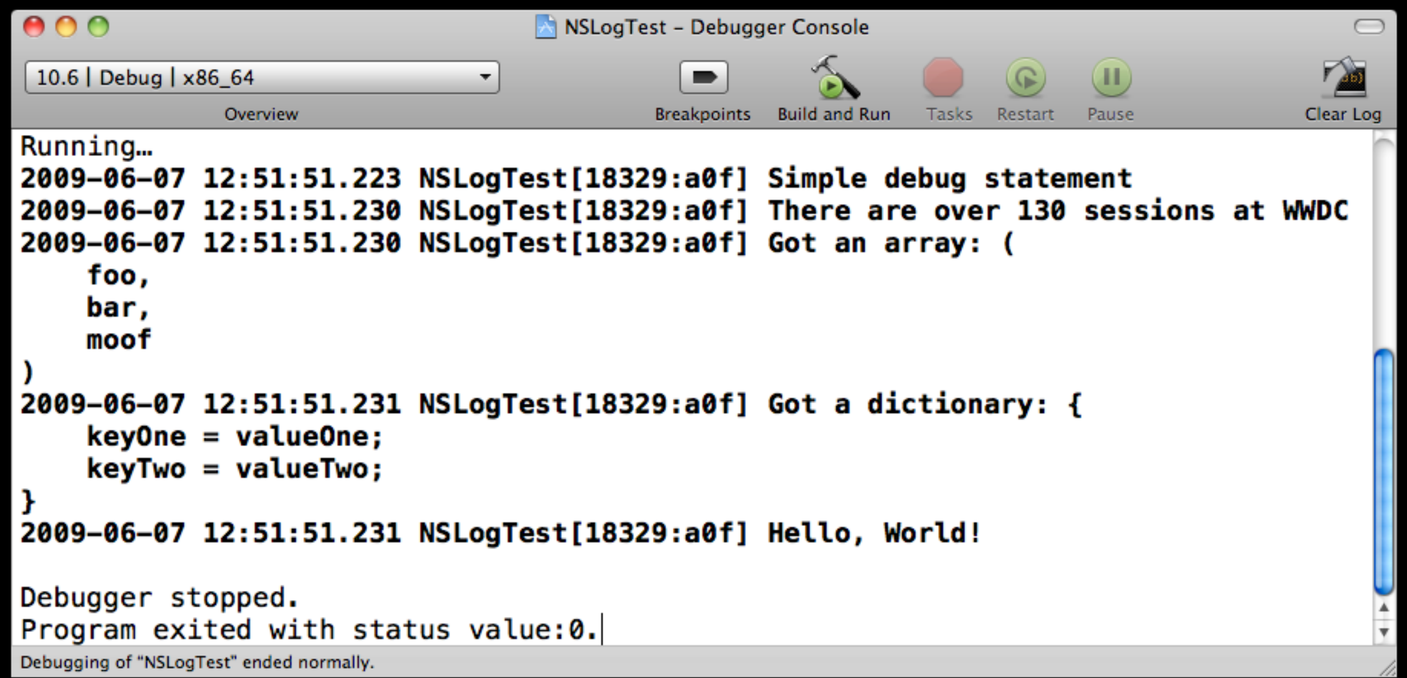
```
NSLog(@"Simple debug statement");

NSLog(@"There are over %d sessions at WWDC", numberOfSessions);

NSLog(@"Got an array: %@", myArray);
NSLog(@"Got a dictionary: %@", myDictionary);
```

Foundation

NSLog

```
NSLog(@"Simple debug statement");

NSLog(@"There are over %d sessions at WWDC", numberOfSessions);

NSLog(@"Got an array: %@", myArray);
NSLog(@"Got a dictionary: %@", myDictionary);
```

```
[myArray description]
```



Demo

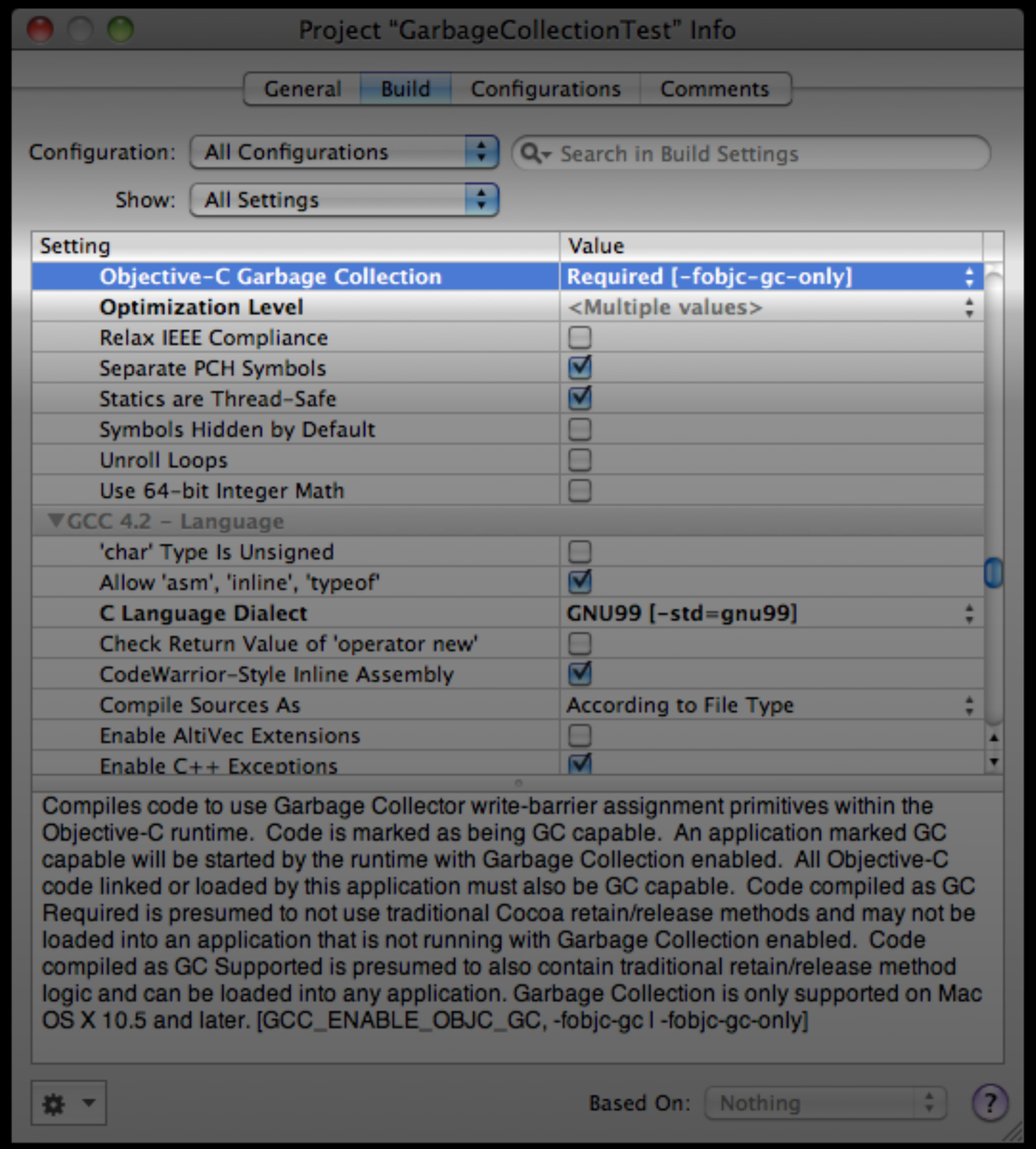
Let’s write something real

Memory Management

Mac Development?

Use Garbage Collection

- Easier
- Faster
- More productive



Reference Counting

A simple system, with simple rules

- Foundation tracks an object's retaining references
- Creating or copying an object often means you own it

```
NSString *owned1 = [NSString new];
NSString *owned2 = [[NSString alloc] init];
NSString *owned3 = [someString copy];
NSString *owned4 = [someString mutableCopy];
```

- No longer need an object you created? -release it!

```
[owned1 release];
```

Reference Counting

A simple system, with simple rules

- Want to keep something around? -retain it!
- Ensure when you're done you -release it

An Example

```
NSArray *foo = [[NSArray alloc] init];
               [foo retain];
              [foo release];
              [foo release];
```

-dealloc

```
+stringWithString:
```

Autorelease Pools

Easy transfer of ownership

- Often used when creating an object and vending it to someone else
- Uses the -autorelease message
- -release will be sent automatically at some point in the future

```
+ (NSString *)stringWithString:(NSString *)input {
    return [[[NSString alloc] initWithString:input] autorelease];
}
```

Autorelease Pools

A delayed release

```
myString = [NSString stringWithString:@”foo”];
```

myString

0

Recap

- If you +alloc'd, +new'd, -copy'd or -retain'd, you must -release
- If you used a convenience method, you must -retain
  - Which of course means you must later -release

Class Design

```
- (id)init {
    if (self = [super init]) {
        element = nil;
        atomicElementView = nil;
        atomicElementFlippedView = nil;
        self.frontViewIsVisible=YES;
        self.hidesBottomBarWhenPushed = YES;
    }
    return self;
}


- (void)loadView {
    // create and store a container view

    UIView *localContainerView = [[UIView alloc] initWithFrame:
   [[UIScreen mainScreen] applicationFrame]];
    self.containerView = localContainerView;
    [localContainerView release];

    containerView.backgroundColor = [UIColor blackColor];
}
```

.h

```
- (id)init {
    if (self = [super init]) {
        element = nil;
        atomicElementView = nil;
        atomicElementFlippedView = nil;
        self.frontViewIsVisible=YES;
        self.hidesBottomBarWhenPushed = YES;
    }
    return self;
}


- (void)loadView {
    // create and store a container view

    UIView *localContainerView = [[UIView alloc] initWithFrame:
   [[UIScreen mainScreen] applicationFrame]];
    self.containerView = localContainerView;
    [localContainerView release];

    containerView.backgroundColor = [UIColor blackColor];
}
```

.m

```
@interface Person : NSObject {
    NSString *name;
    int age;
    NSArray *parents;
}
```

```
  int age;
  NSArray *parents;
}

+ (Person *)personWithName:(NSString *)name;
- (id)initWithName:(NSString *)name;

- (int)age;
- (void)setAge:(int)newAge;

- (NSArray *)parents;
- (void)setParents:(NSArray *)parents;
@end
```

```objc
#import "Person.h"

@implementation Person

    // Your implementation here

@end
```

Initializing Your Instance

```
- (id)init {
    if (self = [super init]) {
        name = @"New Person";
        parents = [[NSMutableArray alloc] init];
    }
    return self;
}
```

Cleaning Up After Ourselves

```
- (void)dealloc {
    [name release];
    [parents release];
    [super dealloc];
}
```

Getting and Setting a Scalar Type

```
- (int)age {
        return age;
}

- (void)setAge:(int)newAge {
        age = newAge;
}
```

Getting and Setting Objects

```
- (NSString *)name {
    return name;
}

- (void)setName:(NSString *)newName {
    if (newName != name) {
        [newName retain];
        [name release];
        name = newName;
    }
}
```

Getting and Setting Objects (GC)

```
- (NSString *)name {
    return name;
}

- (void)setName:(NSString *)newName {
    name = newName;
}
```

Properties

Encapsulation made easy

- Quickly and easily create setter/getter pairs
- Specify storage semantics
- Easy to override for custom behavior

Properties

Interface

- Attributes control storage semantics

```
@property(readwrite) float speed;
```

Attribute	Behavior Controlled
readonly / readwrite	Mutability
assign / retain / copy	Memory management behavior
nonatomic	Concurrency
setter= / getter=	API

Properties

Interface

- Attributes control storage semantics

```
@property(copy) NSArray *passengers;
```

Attribute	Behavior Controlled
readonly / readwrite	Mutability
assign / retain / copy	Memory management behavior
nonatomic	Concurrency
setter= / getter=	API

Properties

Interface

- Attributes control storage semantics

```
@property(nonatomic, retain) NSArray *passengers;
```

Attribute	Behavior Controlled
readonly / readwrite	Mutability
assign / retain / copy	Memory management behavior
nonatomic	Concurrency
setter= / getter=	API

Properties

Interface

- Attributes control storage semantics

```
@property(getter=isGoingFast) BOOL goingFast;
```

Attribute	Behavior Controlled
readonly / readwrite	Mutability
assign / retain / copy	Memory management behavior
nonatomic	Concurrency
setter= / getter=	API

```objc
@interface Person : NSObject {
  NSString *name;
  int age;
  NSArray *parents;
}

- (int)age;
- (void)setAge:(int)newAge;

- (NSArray *)parents;
- (void)setParents:(NSArray *)parents;
@end
```

```objc
@interface Person : NSObject {
  NSString *name;
  int age;
  NSArray *parents;
}

@property(readwrite) int age;

- (NSArray *)parents;
- (void)setParents:(NSArray *)parents;
@end
```

```objc
@interface Person : NSObject {
  NSString *name;
  int age;
  NSArray *parents;
}

@property() int age;

- (NSArray *)parents;
- (void)setParents:(NSArray *)parents;
@end
```

```
@interface Person : NSObject {
  NSString *name;
  int age;
  NSArray *parents;
}

@property() int age;
@property(retain) NSArray *parents;

@end
```

```
@interface Person : NSObject {
  NSString *name;
  int age;
  NSArray *parents;
}

- (int)age;
- (void)setAge:(int)newAge;

- (NSArray *)parents;
- (void)setParents:(NSArray *)parents;
@end
```

```objc
@interface Person : NSObject {
  NSString *name;
  int age;
  NSArray *parents;
}

@property() int age;
@property(retain) NSArray *parents;

@end
```

```
@interface Person : NSObject {
}

@property() int age;
@property(retain) NSArray *parents;
@property(copy) NSString *name;
@end
```

```objc
#import "Person.h"

@implementation Person
@synthesize age, parents, name;

- (id)init {
.....
```

```
- (void)setName:(NSString *)newName {
    if (name != newName) {
        [name release];
        name = [newName mutableCopy];
    }
}
```

Dot Syntax

Easy access to properties

```
Person *bbum = [Person personWithName:@"bbum"];
bbum.age = 35;
```

```
Person *bbum = [Person personWithName:@"bbum"];
NSLog(@"This person's name is: %@", bbum.name);
```

Dynamism in Action

NSString
@"Chordata"

NSString
@"Muuurley"

NSString
@"Charles Drance"

NSString
@"Malonicus"

NSString
@"Trivial"

NSString
@"Johanna"

NSString
@"BBum"

NSString
@"Hopey Hope"

NSString
@"JG"

MJString
- decrypt
- encrypt

NSString
@"Athens"

NSString
@"Blainicus"

NSString
@"Jury"

NSString
@"The funnest"

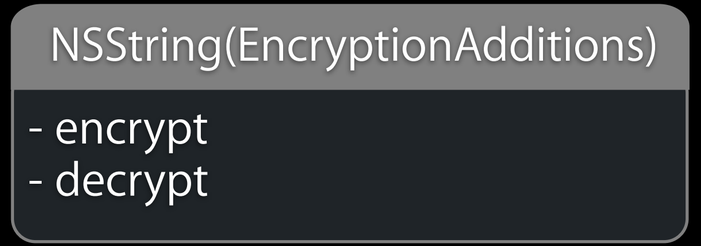
NSString
@"iPhone 3.0"

NSString
@"Snow Leopard"

Categories

Mixing in your own behavior

- Seamlessly add methods to existing classes at runtime



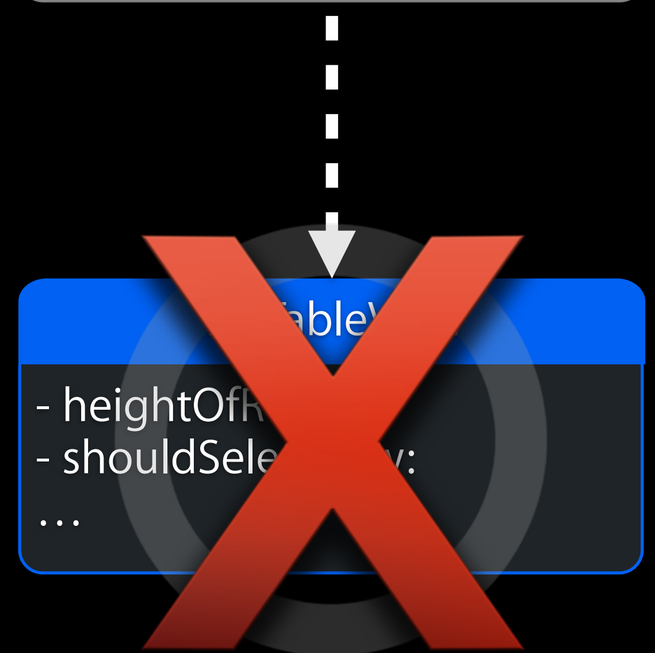
Categories

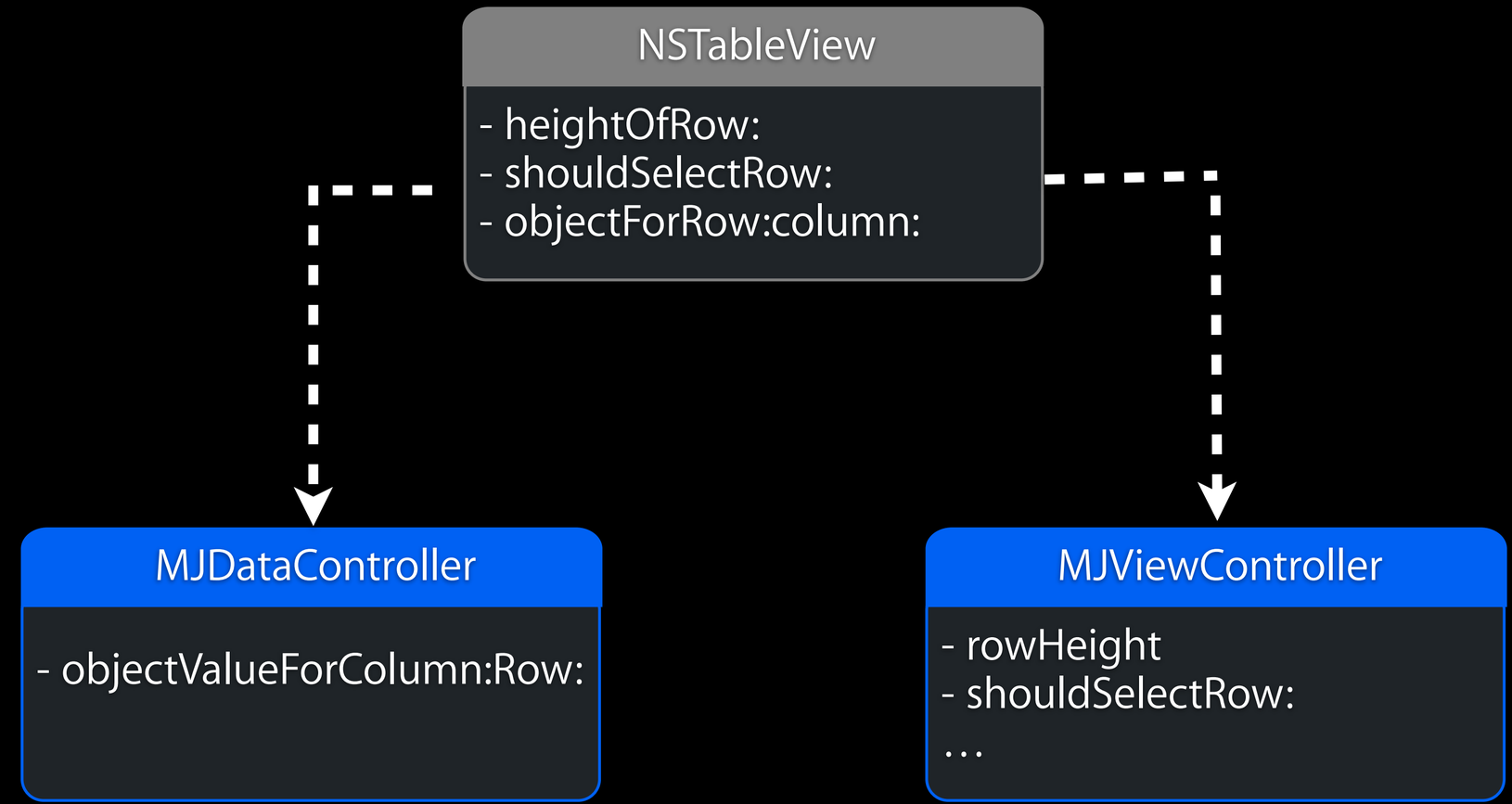
Syntax

```
@implementation NSString (MJEncryptionAdditions)

- (NSString *)MJencrypt {
    ....
}

- (void)MJdecrypt {
   ....
}

@end
```

NSTableView

- heightOfRow:
- shouldSelectRow:
- objectForRow:column:



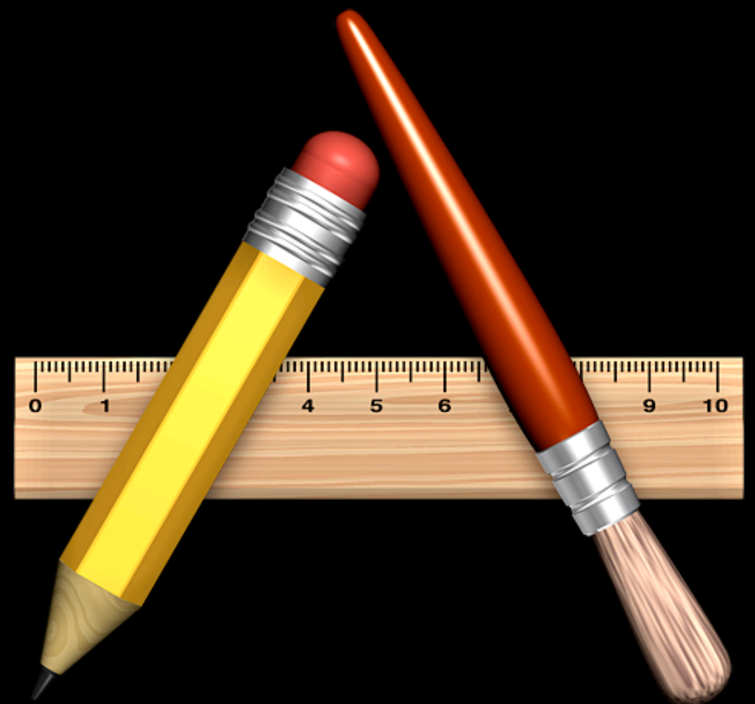

NSTableView
- heightOfRow:
- shouldSelectRow:
- objectForRow:column:
MJDataController
- objectValueForColumn:Row:
MJViewController
- rowHeight
- shouldSelectRow:
…

Delegation

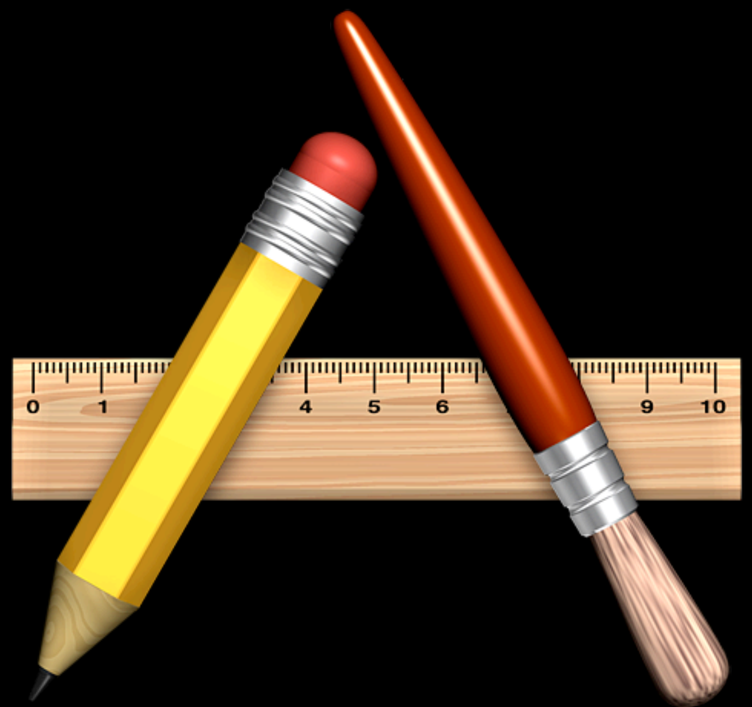
Dynamism in action

- Customize behavior without subclassing
- Provide data
- Customize user interfaces
- Lifecycle notification

Delegation



Delegation



- State changes

```
applicationDidFinishLaunching:
applicationDidHide:
applicationShouldTerminate:
...
```

- Customization and events

```
applicationDockMenu:
application:printFile:
application:willPresentError:
```

Delegation

Cupertino	Blue
San Jose	Red
Mountain View	Purple
Sunnyvale	Yellow
Los Gatos	Green

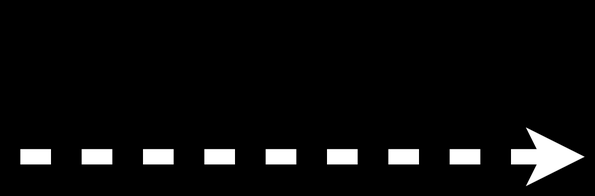
- Data source

```
numberOfRowsInTableView:
tableView:objectValueForTableColumn:row:
...
```

- Delegate

```
tableView:heightOfRow:
tableView:shouldSelectRow:
...
```

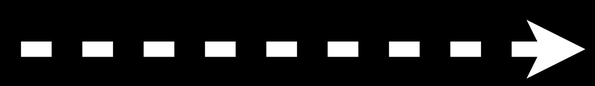
applicationDidHide:



?

Cupertino	Blue
San Jose	Red
Mountain View	Purple
Sunnyvale	Yellow
Los Gatos	Green

tableView:heightOfRow:



?

Protocols

Communication contracts

- A declared list of methods
- Clearly marked as optional and required

```
@protocol UITableViewDataSource <NSObject>
@required
- (NSInteger)tableView:(UITableView *)table
                numberOfRowsInSection:(NSInteger)section;
...

@optional
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
...
@end
```

Protocols

Syntax

```
@protocol UITableViewDataSource <NSObject>
@required
- (NSInteger)tableView:(UITableView *)table
                numberOfRowsInSection:(NSInteger)section;
...

@optional
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
...
@end
```

Protocols

Syntax

```
@protocol UITableViewDataSource <NSObject>
@required
- (NSInteger)tableView:(UITableView *)table
                numberOfRowsInSection:(NSInteger)section;
...

@optional
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
...
@end
```

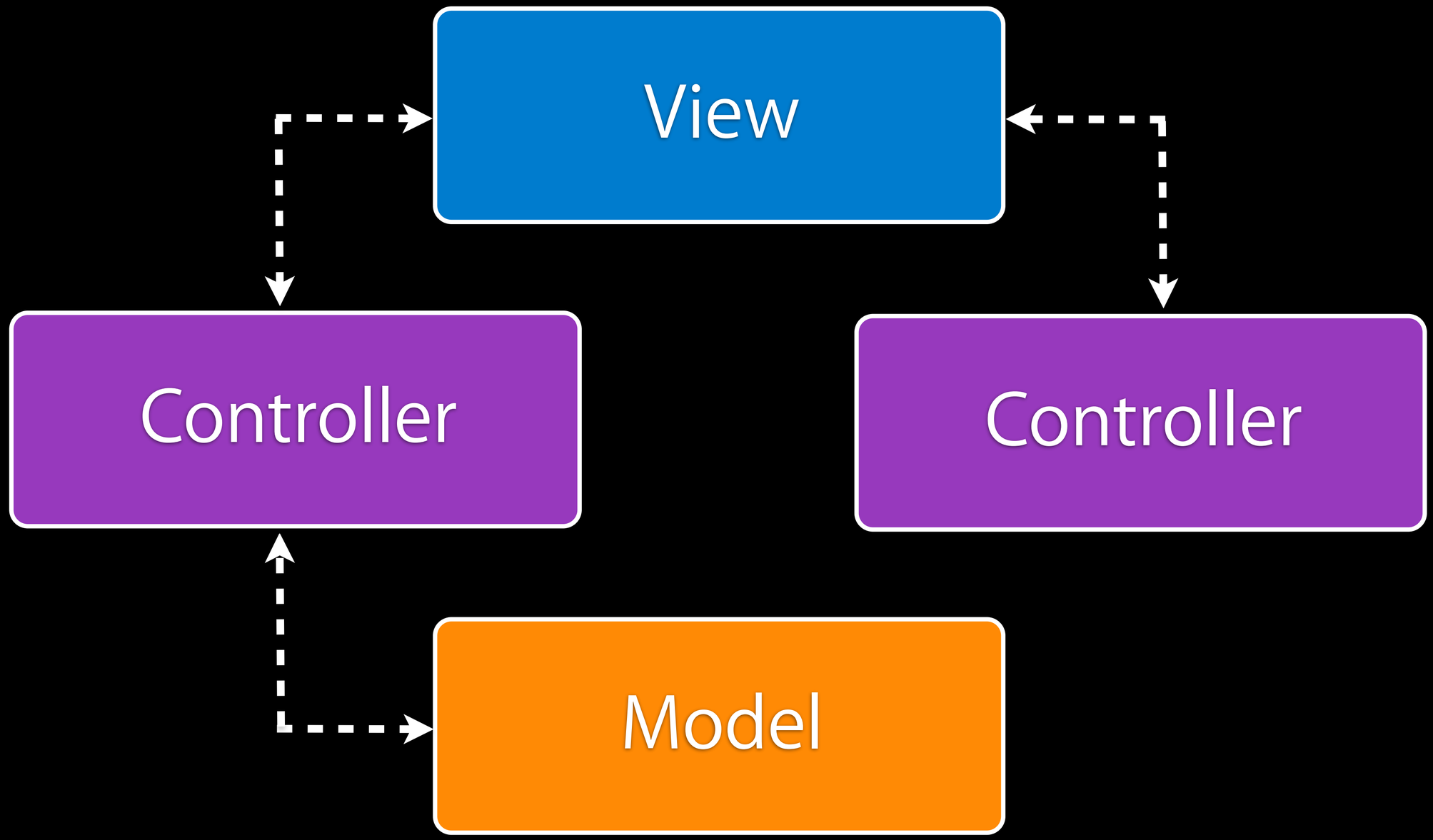
Protocols

Conforming to a protocol

```
@interface DataController : NSObject <UITableViewDataSource> {

}
```

```
@implementation DataController

- (NSInteger)tableView:(UITableView *)table
            numberOfRowsInSection:(NSInteger)section {
   //Your code
}
- (UITableViewCell *)tableView:(UITableView *)tableView
            cellForRowAtIndexPath:(NSIndexPath *)indexPath {
   //Your Code
}
@end
```

Model-View-Controller



What You've Seen

- Practical introduction to Objective-C
- How to design your own classes
- Common design patterns
- Dynamic behavior at the heart of Mac OS X and iPhone OS

Sessions

What's New in Cocoa	Marina Tuesday 9:00AM
What's New in Cocoa Touch	Pacific Heights Tuesday 10:30AM
Objective-C and Garbage Collection Advancements	Russian Hill Wednesday 5:00PM

Labs

Objective-C and Garbage Collection Lab	OS X Lab C Wednesday 9:00AM
Objective-C and Garbage Collection Lab	Snow Leopard Lab Friday 12:00PM

More Information

Michael Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

Documentation

The Objective-C 2.0 Programming Language
http://developer.apple.com

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09